Supplementary Information for *Pycellerator: An arrow-based reaction-like modelling language for biological simulations*

Bruce E Shapiro and Eric Mjolsness

TABLE OF CONTENTS

INSTALLATION

Detailed installation instructions are provided in the user notes, which is the file **pycellerator.pdf**. This file is included as part of the file release on github. This section summarizes the installation instructions in the on-line documentation. For the most recent information subsequent to publication of this paper, users should check the file **readme.txt** or **README.md** on github.

1. Download Pycellerator from the Repository. Pycellerator is available at github. You do not net an account or any special software to download software from github. The download file is located at:

https://github.com/biomathman/pycellerator/releases.

Look for the file:

```
install-pycellerator-v-X.zip
```

(where **X** is some number) and download that file to your computer. Advanced users may be more interested in the source but you don't need that to run to Pycellerator.

2. Unzip and Create Working Folder. You need to unzip the file (which will probably be in your Downloads folder). Look for a folder called **pycellerator** in that unzipped file. Copy this entire folder anywhere you want on your disk drive. This is going to be your working folder for Pycellerator.

3. Install Python 2.7. To use Pycellerator, Python 2.7 and several Python libraries must be installed on your computer. Python is free, open source, and available on all major operating systems. The easiest way to install Python, if you don't already have it, is to install a commercial distribution. Several companies provide free distributions, among them Anaconda (http://continuum.io/downloads) and Entought Canopy (https://store.enthought.com/downloads/). You can also get a binary installer from https://www.Python.org. If you have a Mac, then Python is already installed on your computer. While we summarize the installation instructions here, and they may work on your computer, if you use one of the commercial downloads you should verify the instructions on the product's web site to verify that it is compatible with your operating system and that the installation instructions have not changed since the publication of this document.

4. Install Additional Required Python Libraries. The following additional Python libraries are required for pycellerator: **numpy**, **scypy**, **sympy**, **matplotlib**, **pyparsing**, **pulp**, **libsbml**, and **iPython[notebook]**. The last three are optional, but some features will not be available if they are not installed (flux models, SBML files, or iPython notebooks).

If you are using Anaconda, you can update in the terminal (or Windows Command prompt.).[1] You may not need to upgrade to a more recent version of **pip** (the first line below) but you should try to anyway. If you get a message that says the more recent version is incompatible and can't be installed, that is OK. This means **pip** is already installed. In this case, just continue with the next install.

```
python -m pip install --upgrade pip
pip install pyparsing pulp
conda update conda ipython ipython-notebook
    iPython-qtconsole numpy
    scipy sympy matplotlib
```

Each **conda update** can also be typed as a separate command.

If you are running Canopy, you may need to install **pip**. You may not need to upgrade to a more recent version of **pip** (the first line below), however, but you should try to anyway. If you get a message that says the more recent version is incompatible and can't be installed, that is OK. This means **pip** is already installed. In this case, just continue with the next install.

```
python -m pip install --upgrade pip
pip install --upgrade numpy scipy sympy
     matplotlib pyparsing pulp
```

[1] In all the code blocks, indented lines are used to indicate input that should be typed on a single line of input before hitting the enter key.

Each `pip install` can be typed as a single command

If you install from source, and you are running on Windows, you will require Microsoft Visual C, which you can get from microsoft at:

https://www.Python.org/downloads/windows

Then download the binaries and upgrade using the windows command prompt (`cmd.exe` or `powershell`) with

```
python -m pip install  --upgrade pip
pip install sympy pulp pyparsing setuptools
     numpy scipy matplotlib ipython[notebook]
```

If you have a Mac, then Python is already installed on your computer. The base system is pre-installed as part of the Mac operating system. The base system does not include the numerical libraries that are also required. To upgrade the Mac using `pip`, locate the `terminal` application in the `utilities` folder and open it. Enter

```
sudo easy_install pip
```

When requested, enter you password (you must have administrator access on you Mac). If (when) you are prompted to install `XCode` from Apple, click `yes`, and follow the instructions on any dialog that follows. When the `XCode` installation is completed (or if it was not suggested), open a new terminal session and type in the following. Hit the enter key after each line and wait for the prompt (the name of the current working directory) before typing in the next line.

```
pip install sympy pulp numpy scipy
     pyparsing matplotlib iPython[notebook]
```

Each `pip install` can also be typed as a separate command. If you have a virtual operating system like Parallels Desktop installed on your computer, make sure that Safari (or some other web browser such as Firefox or Chrome) is set as your default browser, and not Parallels. Otherwise iPython will try to open your virtual operating system every time it runs Python.

If you are running on linux, you can install a base Python system from your package manager, download binaries from python.org, build from source (also available at python.org), or use one of the commercial systems. Pycellerator requires Python 2.7.X but is not compatible with Python 3.X. The standard Python installation includes its own package manage called `pip`. If you install the base system from python.org this should automatically be installed for you. Otherwise, you also install `pip` from your package manager. This allows you to bypass your package manager when updating Python. To upgrade to the latest version of `pip`,

```
python -m pip install --upgrade pip
```

To add any missing packages to Python,

```
pip install packagename
```

To upgrade to the latest version of any package,

```
pip install --upgrade package
```

To add the missing pakages, .

```
pip install pyparsing pulp
  sympy numpy scipy matplotlib
```

Binary installers and source code versions are also available for each of these packages.

5. Install libSBML. If you want to work with SBML files, you will also need to install a version of libSBML that is appropriate for your operating system. Make sure to install a version that is compatible with Python.

Follow the instructions at http://sbml.org/Software/libSBML to find the appropriate binary installer for your operating system.

You do not have to have libSBML installed if you do not plan on using SBML files. If libSBML is not installed, all non-SBML related functionality of Pycellerator will be unaffected.

To Install on Ubuntu Linux. If you are primarily interested in using Python with libSBML but not with other languages, this procedure is given by the SBML project. (See http://sourceforge.net/projects/sbml/files/libsbml/5.11.6-experimental/binaries/Linux/)[2] You should check there for the latest information. To install the prerequisites,

```
sudo apt-get install python-dev libxml2-dev
   libz-dev libbz2-dev
```

Then to install libSBML,

```
sudo pip install python-libsbml
```

If you are using Windows, download the latest binary installers can be found at[3]: http://sourceforge.net/projects/sbml/files/libsbml/5.11.4/stable/Windows/64-bit/Python/. These installers will only install libSBML for Python, and not for other languages.

If you are using a mac the latest binaries can be found at[4] http://sourceforge.net/projects/sbml/ files/libsbml/5.11.4/stable/Mac%20OS%20X/. According to the instructions on the repository web page, the installation procedure similar to that for linux.

6. Quick Start Installation Check If you have a complete and correct installation, the following commands should get you up and running immediately with pycellerator.

To verify that the Command Line Interface works, open the terminal from inside the pycellerator folder, and type the following:

[2] As of 5 Sept 2015.
[3] As of 5 Sept 2015.
[4] As of 5 Sept 2015.

```
python pycellerator.py solve -in Gold1.model
   -plot
```

For a quick tutorial of the iPython notebook, type the following into the terminal:

```
iPython notebook
```

Navigate to the notebook **`demo.ipynb`** and open it.

FILE FORMAT

The model is standard text file (e.g., ASCII or UTF-16) that can (and should) be user-modifed with any desired code editor. It consists of a list of text-formatted reactions. An example is shown in Figure S1. Nominally (for easier reading), each reaction should be written on a single line, but multi-line reactions are permitted. The file is divided into multiple sections with special keywords representing the names of the sections. The `$` symbol precedes each keyword. Example keywords are **`$REACTIONS`**, which precede the list of reactions; **`$RATES`** which precedes a list of values of rate constants; **`$IC`**, which preceds a list of intial conditions; **`$FUNCTIONS`**, which precedes a list of user defined functions that are used by the model. The sections may be specified in any order, and the contents of each section may also be specified in any order.

```
$REACTIONS
# -------- Phosphorylation Cascade
 [K3 => K3p, mod[S], rates[a1,d1,k1]]
 [K2 => K2p =>K2pp, mod[K3p], rates[a3,d3,k3]]
 [K => Kp => Kpp, mod[K2pp], rates[a3,d3,k3]]
# -------- Competitive inhibition
 [K3_S + Kpp <-> K3_S_Kpp, rates[a7, d7]]
# -------- Dephosphorylation Cascade
 [K3p => K3, mod[K3PH], rates[a4,d4,k4]]
 [K2pp => K2p => K2, mod[K2PH], rates[a5,d5,k5]]
 [Kpp => Kp => K, mod[KPH], rates[a6,d6,k6]]
# --------  Stimulation
 [Nil<->S, rates["a0*fon(t)*foff(t)", d0]]
$Functions
 foff(t)=(1-.5*(1+(t-stim_stop)/(1+(t-stim_stop)**2)**.5))
 fon(t)=0.5+0.5*(1+t-stim_start)/(1+(t-stim_start)**2)**.5
$IC
 S=1;   K3=100; K3p=0; K2=300; K2p=0;   K2pp=0
 K=300; Kp=0;   Kpp=0; KPH =1; K2PH =1; K3PH=10
$Rates
 a0=1;    a1=1;     a3=1; a5=1; a4=1; a6=1; a7=1;
 d0=0.01; d1=7.5;   d3=10;d4=1; d5=1; d6=1; d7=1;
 k1=2.5;  k3=0.025; k4=1; k5=1; k6=1;
 t_start=750; t_stop=6000
```

Fig. S1. Sample input file. Sections are delimited with the "`$`", lines may be separated with either newlines or semicolons, and comments begin with "#". The functions $f_{\text{on}}(t) = (1/2)(1 + (1 + t - t_{\text{start}})/(\sqrt{1 + (t - t_{\text{start}})^2}))$ and $f_{\text{off}}(t) = 1 - (1/2)(1 + (t - t_{\text{stop}})/\sqrt{1 + (t - t_{\text{stop}})^2}$ are sigmoidal functions that turn the stimulation on at t_{on} and off at t_{off}, approximating a differentiable step function.

WORK FLOW

The standard way of representing a model is by a model file, but models may also be represented by either legacy (Mathematica) Cellerator files, or as SBML model files. As illustrated in figure S2, built in functions will convert these files to readable model files. Once a model file has been created, it can be

- Read and converted to a system of differential equations (interpreter module);
- Read and converted to an ODE-based Python computer program (solver module);
- Read and converted to a Flux-based Python computer program (flux module, not shown); flux models and ODE models are mutually exclusive;
- The code may be run (solver module);
- Time courses may be plotted.
- Parametric scans may be run and plotted.

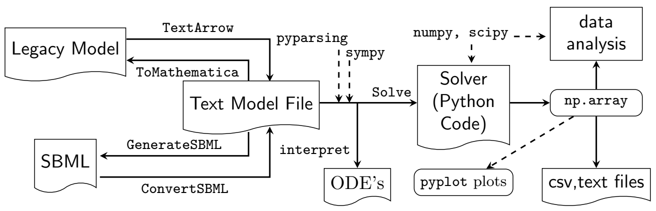

Fig. S2. Overall work flow for Cellerator using iPython.

STANDARD CELLERATOR ARROWS

All functionality is preserved in the Python implementation, however, the syntax has been changed. The major changes are summarized as follows.

- Arrows are all expressible in standard typewriter characters (e.g., ASCII) so that an interpreter (such as Mathematica) is not required to read and translate the model file. For the most part, the correspondences are as given in table S1.
- The use of overscripts and underscripts has been removed in the new format. In the legacy format modifier species, species whose concentrations are not affected by a reaction, but whose presence affect the outcome of a reaction in their kinetic law, were written either above or below the reaction arrow, as in

$$\{\mathtt{A} \overset{\mathtt{X}}{\rightleftarrows} \mathtt{B}, \mathtt{k1}, \mathtt{k2}, \mathtt{k3}, \mathtt{k4}\} \tag{1}$$

  They are now moved within a function **`mod`**, as in

$$[\mathtt{A} => \mathtt{B}, \mathtt{mod}[\mathtt{X}], \mathtt{rates}[\mathtt{k1}, \mathtt{k2}, \mathtt{k3}, \mathtt{k4}]] \tag{2}$$

  Reactions (1) and (2) represent the same reaction in the old and new formats, respectively.

- Rate constants are encapsulated within a function **`rates`** at the end of the arrow, as in reaction (2).
- The usage of curly braces (as in{**`expression`**}) in the Mathematica form has been replaced with square brackets as in **`[expression]`**) to maintain consistency with the Python list implementation.

Supplementary Table S2 illustrates the syntactical differences between mass action arrows as implemented in the Mathematica code and in the Python code. The differences in other arrows are summarized in Supplementary Table S3.

Table S1. Comparison of text and legacy arrows.

Legacy	Text	Typical usages
$\rightarrow$	`->`	Mass action, User defined
$\rightarrow$	`-->`	Catalytic Mass action
$\leftrightarrow$	`<->`	Bi-directional mass action
$\rightleftarrows$	`=>`	Enzymatic with intermediate complex formation
$\rightleftharpoons$	`:=>`	Enzymatic with two intermediate complexes
$\rightleftarrows$	`<=>`	Bi-directinal enzymatic with single intermediate complex formation
$\mapsto$	`|->`	Hill, GRN, S-System, NHCA, User
$\mapsto$	`|-->`	Catalyzed Hill, GRN, S-System, User
$\Rightarrow$	`:->`	Michaeles-Menten-Henri (MMH)
$\Rightarrow$	`:-->`	Catalyzed MMH
$\Longrightarrow$	`==>`	MWC, Rational

CASCADES

Any mass action, catalytic mass action, MMH, Hill Function, GRN, S-System, or NHCA reaction can be written in a cascade as a single reaction. A cascade is defined as sequence of repeated reactions with the same arrow and the same rate constants. For example, the reactions

```
[A => B, mod[E], rates[k1,k2,k3]]
[B => C, mod[F], rates[k1,k2,k3]]
[C => D, mod[G], rates[k1,k2,k3]]
```

can be written as a single reaction cascade:

```
[A => B => C => D, mod[E, F, G],
   rates[k1,k2,k3]]
```

Reactions without modifiers can also be written as cascades:

```
[P :-> Q :-> R, MMH[KD, v]]
```

which represents the pair of reactions

```
[P :-> Q, MMH[KD, v]]
[Q :-> R, MMH[KD, v]]
```

USER DEFINED ARROWS

Two types of user defined reactions are possible. The first user-defined reaction is expressed as

```
[e_1 * X_1 + e_2 * X_2 + ··· ->
   f_1 * Y_1 + f_2 * Y_2 + ··· using[expr]]
```

where **`expr`** is any valid Python expression enclosed in quotes. (Note that the subscripts are not valid in Python and are merely used here for illustrative purposes.) The second type of user defined reaction is

```
[X_1 + X_2 + ··· |->Y,
  USER[v, [T_1, T_2, ...], [n_1, n_2, ...], h, f]]
```

where **`f`** is a function defined in the model and the differential equation term is computed as

$$\frac{dY}{dt} = vf(h - \sum T_i X_i^{n_i}) \tag{3}$$

REPRESENTING FLUXES

A flux arrow is represented by the form

```
[ LHS -> RHS, Flux[low < id < up, obj, flux]
```

The formats are analogous to COBRA fluxes: **`low`** and **`hi`** correspond to upper and lower bounds; **`id`** is an identifier in the model; **`obj`** is an objective coefficient; and **`flux`** is a flux value. Equality constraints are obtained by setting **`low`** and **`hi`** to the same value. Flux optimization maximizes the dot product $\mathbf{v}^T\mathbf{f}$ subject to $\mathbf{Nv} = \mathbf{0}$ and all the supplied constraints, where $\mathbf{v}$ is the vector of fluxes, $\mathbf{N}$ is the stoichiometry matrix, and $\mathbf{f}$ is the vector of objective coefficients. A model may be composed either entirely of standard arrows or entirely of flux arrows, but they may not be combined.

SOFTWARE DEPENDENCIES

Cellerator utilizes standard Python 2.7 libraries. It assumes that the following additional Python libraries are also installed: numpy, scipy (numerical and scientific Python libraries), pyparsing (a BNF based parser), and matplotlib (plotting library). To process flux models, the pulp (linear programming) library is required, and to read or write SBML models, libSBML is required. If either pulp or libsbml are not available the rest of the program should still work. In addition to a full iPython notebook installation, if you wish to

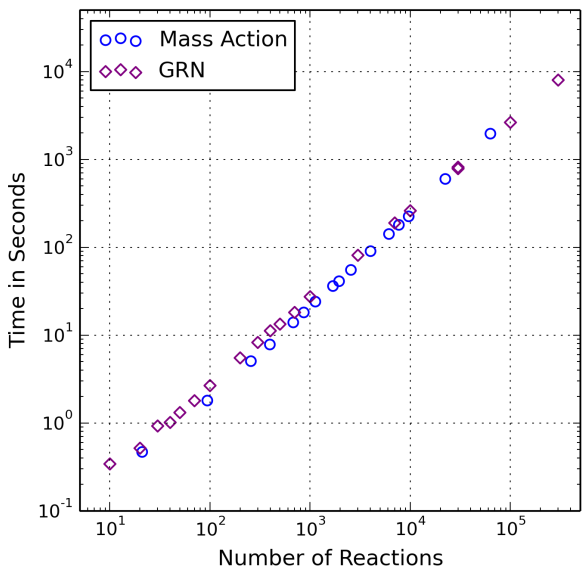

Fig. S3. Scaling of the code generation algorithm. The markers show the time taken to generate (and save to disk) stand-alone Python solvers from randomly generated models of the given size.

export iPython notebooks to other formats such as html or LaTeX, the Python pandoc document conversion library is also required. The stand-alone code generated by Cellerator depends only on numpy and scipy.

SCALING OF CODE GENERATION ALGORITHM

The automated code generation algorithm scales linearly with the number of reactions. This was determined by generating random models (with random reactions and random values for rate constants) containing up to approximately 300,000 reactions ad measuring the compute time on an Intel Core i7-3770K CPU with 32MB memory running Ubuntu 14.10. Models contained either mass-action reactions or GRN reactions. Mass-action reactions were randomly generated with the form

$$X_i + X_j \overset{k_q}{\to} \sum_{p \in P_q} X_p \tag{4}$$

so that every species interacted, on average, with approximately 50% of the other species in any given model. The set of products P_q was randomly chosen for each reaction, with a maximum size of 5 species. The GRN reactions were also randomly generated, and of the form

$$\sum_{w \in W_q} X_w \mapsto X_v \tag{5}$$

where W_q would contain up to seven species, randomly chosen for each reaction, e.g., representing various various transcription factors or upstream proteins that control the transcription of X_v. All rate constants (k_q for the mass action reactions, and the T, h, and exponent values for the GRN) were randomly assigned. The results show a linear scaling over the range of model sizes tested, as illustrated in fig. S3.

GLOSSARY

ASCII - American Standard Code for Information Interchange, an old seven bit text-encoding format derived from telegraphic standards, which has generally been replaced by UTF-8.

API - Application Program Interface. In this case, it refers to a set of function calls that can be used to access the individual low level capabilities of Pycellerator directly via calls from other Python programs.

BDF - Backward Differentiation Formula. One of the families of techniques in which a differential equation is solved numerically. It solves the differential equation $y' = f(t, y)$ (y is a vector) by solving a system of linear equations

$$\sum_{k=0}^{N} a_k y_{n+k} = hAf(t_{n+N}, y_{n+N})$$

for some fixed step size h, order N, and constants a_0,.., a_N and A. for the y_j. Since the equations are implicit in y_j, an iterative solver is usually used.

BNF - Backus-Naur Form or Backus-Normal Form, a notation used to describe context free grammars. Both names are used; Peter Naur invented the second, while Donald Knuth invented the first. BNF is a meta-language used to describe computer languages. the typical syntax, e.g., to describe the addition operation **`A+B`** as a single term or the sum of terms containing integers or symbols,

```
<term>       ::<integer>|<symbol>
<expression>::=term>|<expression>"+"<term>
<integer>    :: ... etc
```

Cellerator - (1) **`Cellerator`** is a Mathematica package designed to facilitate biological modeling via automated equation generation. **`Cellerator`** was designed with the intent of simulating at least the following essential biological processes: signal transduction networks (STNs); cells that are represented by interacting signal transduction networks; and multi-cellular tissues that are represented by interacting networks of cells that may themselves contain internal STNs. See http://bioinformatics.oxfordjournals.org/content/19/5/677.abstract. (2) **`Cellerator`** is also the name of a Python include file that is part of the iPython interface for Pycellerator, which acts as a high-level interface between iPython and the Pycellerator libraries.

Cellzilla - **`Cellzilla`** is a two-dimensional tissue simulation platform for plant modeling utilizing Cellerator arrows. See http://journal.frontiersin.org/article/ 10.3389/ fpls.2013.00408/

CLI - Command Line Interface. In this case, it refers to the ability to execute Pycellerator via commands typed into the terminal application (in linux or Mac OS; called the Command Prompt in Windows, cmd.exe, or powershell).

Conda - `conda` is a software management system for Python. It is used by the Anaconda Python system. It lets you install, update and configure software under the Anaconda Python distribution. For more information see http://conda.pydata.org/docs/.

eval - `eval` is a Python function that evaluates a string as if it were a line of code.

GPL - Acronym for GNU General Public License, a copyleft software license that allows users the freedoms to run, study, share (copy), and modify the software subject to the conditions of the license. For more information see http://www.gnu.org/licenses/gpl .html.

GRN - Acronym for Gene Regulatory Network reaction. In Pycellerator, this refers to a reaction $A + B + C + \cdots \to X$ in which the concentrations of A, B, and C are not affected, but in which X changes by the logistic function

$$X' = \frac{v}{1 + e^{-h-\beta_1 A_1^n - \beta_2 B_2^n - \beta_3 C_3^n - \cdots}}$$

for some fixed constants h, v, β_1, β_2,.. , n_1, n_2,..

iPython - an interactive computational environment, in which you can combine code (Python), code output, markdown, and graphics. For more information see http://iPython.org/notebook.html

Jupyter - A web application designed to share visualizations. It is used by the iPython notebook environment. For more information see https://jupyter.org/

KMech - an enzyme mechanism language for the mathematical modeling of metabolic pathways, implemented in Mathematica using Cellerator. It has been completely re-implemented in the present paper as part of Pycellerator. For more information on the original KMech see http://bioinformatics.oxfordjournals.org/content /21/6/774.abstract

LSODA - A library to solve differential equations. LSODA solves systems $dy/dt = f$ automatically selects between nonstiff (Adams) and stiff (BDF) methods, using a nonstiff method initially, and dynamically monitoring data to decide which method to use. For more information see https://people.sc.fsu.edu/ jburkardt/f77_src /odepack/odepack.html

matplotlib - A 2D plotting library for Python. For more information see http://matplotlib.org

MMH - An acronym for Michaelis-Menten-Henri Kinetics. Catalyzed conversion of a substrate S into a product P, described by, e.g.,

$$\mathrm{S} + \mathrm{E} \underset{k_2}{\overset{k_1}{\rightleftarrows}} \mathrm{SE} \overset{k_3}{\rightarrow} \mathrm{P} + \mathrm{E}$$

Under certain conditions, the rate of the reaction is

$$\mathrm{P}' = \frac{k_3 \mathrm{A}}{K + \mathrm{A}}$$

where

$$K = \frac{k_2 + k_3}{k_1}$$

MWC - Acronym for Monod-Wyman-Changeaux model for allosteric transitions. For more details see http://www.worldscientifi c.com/doi/abs/ 10.1142/S0219720006001862

numpy - `numpy` is the standard numerical computing package for Python. For more information see http://www.numpy.org/

odeint - `scipy.integrate.odeint` is the standard solver for differential equations in Python. For more information see http://docs.scipy.org/doc/scipy/reference/generated/scipy. integrate. odeint.html

pip - `pip` is the Python package manager. It is used to install, update, or remove parts of your Python installation.

pulp - `pulp` is a linear programming library for Python. For more information see https://github.com/coin-or/pulp

pyparsing - `pyparsing` is a parsing library for Python. http://sourcef orge.net/projects/pyparsing/

SBML - An acronym for Systems Biology Modeling Language: "a free and open interchange format for computer models of biological processes. SBML is useful for models of metabolism, cell signaling, and more." http://sbml.org

scipy - `scipy` is a collection of Python libraries, including one named `scipy`, for scientific computing in Python. Other libraries used by Pycellerator, such as `numpy` and `matplotlib`, are considered part of the "scipy stack." See http://www.scipy.org/

sudo - `sudo` is a unix-like command that means "do like `su`", where `su` is short for "superuser" or administrator. It gives the command administrator privilege.

sympy - `sympy` is a Python library for computer algebra. More details are at http://www.sympy.org/en/

UTF-16 - A standard 16 bit character encoding for text files.

Table S2. Comparison of Python text arrow formats with legacy arrow formats as displayed in the Mathematica notebook front end processor.

	Cellerator Arrow in Mathematica Notebook	Python Text Format[a,b]	Typical Biochemical	Typical ODE Term or Expansion
(1)	$\{\mathtt{X \rightarrow Y,\ k}\}$	`[X -> Y, k]`	$\mathrm{X} \xrightarrow{k} \mathrm{Y}$	$Y' = -X' = kX$
(2)	$\{\mathtt{e_1X_1+e_2X_2+\cdots \rightarrow}$ $\mathtt{f_1Y_1+f_2Y_2+\cdots,\ k}\}$	$\mathtt{[e_1{*}X_1+e_2{*}X_2+\cdots{-}{>}}$ $\mathtt{f_1{*}Y_1+s_2{*}Y_2+\cdots,k]}$	$\sum_i e_i\mathrm{X}_i \xrightarrow{k} \sum_j f_j\mathrm{Y}_j$	$U'_m = k(f_{U_m} - e_{U_m})\prod_{lhs} X_i^{e_i}$
(3)	$\{\mathtt{e_1X_1+e_2X_2} + \cdots \xrightarrow{\mathtt{M}}$ $\mathtt{f_1Y_1+f_2Y_2\ +\ \cdots,\ k}\}$	$\mathtt{[e_1X_1+e_2X_2+\cdots{-}{-}{>}}$ $\mathtt{f_1Y_1+f_2Y_2+\cdots,}$ `mod[M], k]`	$\mathrm{M} + \sum_i e_i\mathrm{X}_i \xrightarrow{k}$ $\mathrm{M} + \sum_j f_j\mathrm{Y}_j$	$U'_m = kM(f_{U_m} - e_{U_m})\prod_{lhs} X_i^{e_i}$
(4)	$\{\mathtt{e_1X_1+e_2X_2+\cdots} \rightleftarrows$ $\mathtt{f_1Y_1+f_2Y_2+\ \cdots,}$ $\mathtt{k_1,\ k_2}\}$	$\mathtt{[e_1X_1+e_2X_2+\cdots{<}{-}{>}}$ $\mathtt{f_1Y_1+f_2Y_2+\cdots,}$ $\mathtt{rates[k_1,k_2]]}$	$\sum_i e_i\mathrm{X}_i \underset{k2}{\overset{k_1}{\rightleftarrows}} \sum_j f_j\mathrm{Y}_j$	Expands to two arrows of the form (2)
(5)	$\{\mathtt{S} \overset{\mathcal{E}}{\rightleftarrows} \mathtt{P,}$ $\mathtt{k_1,k_2,k_3,k_4}\ \}$	`[S=>P,mod[E],` $\mathtt{rates[k_1,k_2,}$ $\mathtt{k_3,k_4]]}$	$\mathrm{S+E} \underset{k_2}{\overset{k_1}{\rightleftarrows}} \mathrm{SE} \underset{k_4}{\overset{k_3}{\rightleftarrows}} \mathrm{P+E}$ or: $\mathrm{S+E} \xrightarrow{k_1} \mathrm{SE}$ $\mathrm{SE} \xrightarrow{k_2} \mathrm{S+E}$ $\mathrm{SE} \xrightarrow{k_3} \mathrm{P+E}$ $\mathrm{P+E} \xrightarrow{k_4} \mathrm{PE}$	Expands into four arrows: $\mathtt{[S{+}E{-}{>}S\_E,k_1]}$ $\mathtt{[S\_E{-}{>}S{+}E,k_2]}$ $\mathtt{[S\_E{-}{>}P{+}E,k_3]}$ $\mathtt{[P{+}E{-}{>}S\_E,k_4]}$

(6)	$\{\mathtt{S} \underset{\mathcal{R}}{\overset{\mathcal{F}}{\rightleftarrows}} \mathtt{P,}$
	$\mathtt{k_1,k_2,k_3,k_4,}$
	$\mathtt{k_5,k_6,k_7,k_8}\}$
		`[S<=>P,mod[F,R],`
		$\mathtt{rates[k_1,k_2,}$
		$\mathtt{k_3,k_4,k_5,}$
		$\mathtt{k_6,k_7,k_8]]}$
			$\mathrm{S+F} \underset{k_2}{\overset{k_1}{\rightleftarrows}} \mathrm{SF} \underset{k_4}{\overset{k_3}{\rightleftarrows}} \mathrm{P+F}$
			$\mathrm{P+R} \underset{k_6}{\overset{k_5}{\rightleftarrows}} \mathrm{PR} \underset{k_8}{\overset{k_7}{\rightleftarrows}} \mathrm{S+R}$
			or:
			$\mathrm{S+F} \xrightarrow{k_1} \mathrm{SF}$ $\quad$ $\mathrm{SF} \xrightarrow{k_2} \mathrm{S+F}$
			$\mathrm{SF} \xrightarrow{k_3} \mathrm{P+F}$ $\quad$ $\mathrm{P{+}F} \xrightarrow{k_4} \mathrm{SF}$
			$\mathrm{P+R} \xrightarrow{k_5} \mathrm{PR}$ $\quad$ $\mathrm{PR} \xrightarrow{k_6} \mathrm{P+R}$
			$\mathrm{PR} \xrightarrow{k_7} \mathrm{S+R}$ $\quad$ $\mathrm{S{+}R} \xrightarrow{k_8} \mathrm{SR}$
				Expands into two arrows of type (5) thence 8 arrows type (2):
				`[S=>P, mod[F],`
				$\mathtt{rates[k_1,k_2,k_3,k_4]]}$
				`[P=>S, mod[R],`
				$\mathtt{rates[k_5,k_6,k_7,k_8]]}$
				and thence:
				$\mathtt{[S{+}F{-}{>}S\_F,k_1],\ [S\_F{-}{>}S{+}F,k_2]}$
				$\mathtt{[S\_F{-}{>}P{+}F,k_3],\ [P{+}F{-}{>}S\_F,k_4]}$
				$\mathtt{[P{+}R{-}{>}P\_R,k_5],\ [P\_R{-}{>}P{+}R,k_6]}$
				$\mathtt{[P\_R{-}{>}S{+}R,k_7],\ [S{+}R{-}{>}P\_R,k_8]}$

(7)	$\{\mathtt{S} \overset{\mathcal{E}}{\rightleftharpoons} \mathtt{P,}$ $\mathtt{k_1,k_2,k_3,}$ $\mathtt{k_4,k_5,k_6}\ \}$	`S:=>P,mod[E],` $\mathtt{rates[k_1,k_2,k_3,}$ $\mathtt{k_4,k_5,k_6]]}$	$\mathrm{S+E} \underset{k_2}{\overset{k_1}{\rightleftarrows}} \mathrm{SE} \underset{k_4}{\overset{k_3}{\rightleftarrows}} \mathrm{PE} \underset{k_6}{\overset{k_5}{\rightleftarrows}} \mathrm{P+E}$ or: $\mathrm{S+E} \xrightarrow{k_1} \mathrm{SE}$ $\quad$ $\mathrm{SE} \xrightarrow{k_2} \mathrm{S+E}$ $\mathrm{SE} \xrightarrow{k_3} \mathrm{PE}$ $\quad$ $\mathrm{PE} \xrightarrow{k_4} \mathrm{SE}$ $\mathrm{PE} \xrightarrow{k_5} \mathrm{P+E}$ $\quad$ $\mathrm{P+E} \xrightarrow{k_6} \mathrm{PE}$	Expands to 6 arrows type (2): $\mathtt{[S{+}E{-}{>}S\_E,k_1]\quad [S\_E{-}{>}S{+}E,k_2]}$ $\mathtt{[S\_E{-}{>}P\_E\ k_3]\quad [P\_E{-}{>}S\_E,k_4]}$ $\mathtt{[P\_E{-}{>}P{+}E,k_5]\quad [P{+}E{-}{>}P\_E,k_6]}$

[a] Subscripts are for illustrative purposes only; subscripts are not possible in the text format. [b] The asterisk ($\star$) between stoichiometry and species is not required when the stoichiometry is a literal number.

Table S3. Comparison of Cellerator arrows in Python text format and legacy mathematica format.

Legacy Arrows	Text Arrows	ODE Terms
	Michaelis-Menten-Henri Type Reactions (MMH)	
$\{$**A⇒B,MM[Kv]**$\}$ $\{$**A⇒B,MM[**k_1**,**k_2**,**k_3**]**$\}$ $\{$**A**$\overset{X}{\Rightarrow}$**B,MM[K,v]**$\}$ $\{$**A**$\overset{X}{\Rightarrow}$**B,MM[**k_1**,**k_2**,**k_3**]**$\}$	**[A:->B,MMH[K, v]]** **[A:->B,MMH[**k_1**,**k_2**,**k_3**]]** **[A:->B,mod[X],MMH[K, v]]** **[A:->B,mod[X],MMH[**k_1**,**k_2**,**k_3**]]**	$B' = -A' = \frac{vA}{K+A}$ $B' = -A' = \frac{k_3A}{(k_2+k_3)/k_1+A}$ $B' = -A' = \frac{vAX}{K+A}$ $B' = -A' = \frac{k_3AX}{(k_2+k_3)/k_1+A}$
	Hill Functions	
$\{$**A↦B,Hill[v,n,K,a,T]**$\}$ $\{\{$**P**$_1$**,P**$_2$**,..**$\}$**↦Q,Hill[v,n,K,a,** $\{$**T**$_1$**,T**$_2$**,..**$\}$**]**$\}$ $\{\{$**X**$_1$**,X**$_2$**,..**$\}\overset{\mathcal{E}}{\mapsto}$**Y,Hill[v,n,K,a,** $\{$**T**$_1$**,T**$_2$**,..**$\}$**]**$\}$	**[A|->B,Hill[v,n,K,a,T]]** **[[P**$_1$**,P**$_2$**,..]|->Q,Hill[v,n,K,a,** **[T**$_1$**,T**$_2$**,..]]]** **[[X**$_1$**,X**$_2$**,..]|-->Y,mod[E],Hill[v,** **n,K,a,[T**$_1$**,T**$_2$**,..]]]**	$B' = \frac{v(a+TA)^n}{K^n+(a+TA)^n}; A'=0$ $Q' = \frac{v(a+\sum T_jP_j)^n}{K^n+(a+\sum T_jP_j)^n}; P'_j = 0$ $Y' = \frac{v\mathcal{E}(a+\sum T_jX_j)^n}{K^n+(a+\sum T_jX_j)^n} = -X'_i$
	Gene Regulatory Network (GRN, Logistic) Arrows	
$\{$**A↦B,GRN[v,T,n,h]**$\}$ $\{\{$**P**$_1$**,P**$_2$**,..**$\}$**↦Q,GRN[v,**$\{$**T**$_1$**,..**$\}$**,n,h]**$\}$ $\{\{$**X**$_1$**,X**$_2$**,..**$\}\overset{\mathcal{E}}{\mapsto}$**Y,GRN[v,**$\{$**T**$_1$**,..**$\}$**,n,h]**$\}$	**[A|->B,GRN[v,T,n,h]]** **[[P**$_1$**,..]|->Q,GRN[v,[T,..],n,h]]** **[[X**$_1$**,..]|-->Y,mod[E],GRN[v,** **[T,..],n,h]]**	$B' = \frac{v}{1+e^{-(h+TA^n)}}; A'=0$ $Q' = \frac{v}{1+e^{-(h+\sum T_jP_j^n)}}; P'_i = 0$ $Y' = \frac{v\mathcal{E}}{1+e^{-(h+\sum T_jX_j^n)}}; X'_i = 0$
	S-Systems	
$\{\{$**S**$_1$**,S**$_2$**,..**$\}$**↦P,SSystem[**τ**,a,b,** $\{$**g**$_1$**,g**$_2$**,..**$\}$**,**$\{$**h**$_1$**,..**$\}$**]**$\}$	**[[S**$_1$**,..]|->P,SSystem[tau,a,b,** **[g**$_1$**,..],[h**$_1$**,..]]**	$P' = \frac{1}{\tau}\left(a\prod_i S_i^{g_i} - b\prod_i S_i^{h_i}\right)$
	Rational Functions	
$\{\{\{$**X**$_1$**,X**$_2$**,..**$\}$**,**$\{$**Y**$_1$**,Y**$_2$**,..**$\}\} \Longrightarrow$ **Z,** **rational[**$\{$**a**$_0$**,..**$\}$**,**$\{$**d**$_0$**,..**$\}$**,** $\{$**m**$_1$**,..**$\}$**,**$\{$**n**$_1$**,..**$\}$**]**$\}$	**[[[X**$_1$**,..],[Y**$_1$**,..]]==>Z,rational[** **[a,..],[d,..],[m,..],[n,..]]]** **[[[X**$_{11}$***X**$_{12} * \cdots$**,X**$_{21}$***X**$_{22} * \cdots$**,..],** **[Y**$_{11}$***Y**$_{12}$*****$\cdots$**,Y**$_{21}$***Y**$_{22}$*****$\cdots$**,..]]** **==>Z,rational[[a,..],[d,..],** **[m,..],[n,..]]]**	$Z' = \frac{a_0^{m_0}+\sum_i a_iX_i^{m_i}}{d_0^{m_0}+\sum_i d_iY_i^{n_i}}$ $Z' = \frac{a_0^{m_0}+\sum_i a_i(X_{i1}X_{i2}\cdots)^{m_i}}{d_0^{m_0}+\sum_i d_i(Y_{i1}Y_{i2}\cdots)^{n_i}}$
	Monod-Wyman-Changeaux Type Reactions (MWC)	
$\{$**S** $\overset{\mathcal{E}}{\Longrightarrow}$ **P,MWC[k,n,c,L,K]**$\}$ $\{$**S** $\underset{\{\{A_1,..\},\{I_1,..\}\}}{\overset{\mathcal{E}}{\Longrightarrow}}$ **P,MWC[k,n,c,L,K]**$\}$ $\{$**S** $\underset{\{\{A_1,.\},\{I_1,..\},\{\{C_1,.\},.\},.\}}{\overset{\mathcal{E}}{\Longrightarrow}}$ **P,MWC[k,n,c,L,K,..]**$\}$	**[S==>P,mod[E],MWC[k,n,c,L,K]]** **[[S**$_1$**..]==>P,mod[E,[A**$_1$**..],[I**$_1$**,..]],** **MWC[k,n,c,L,K]]** **[[S**$_1$**,..]==>P,mod[E,[A**$_1$**,..],[I**$_1$**,..],** **[[C**$_1$**],..]], MWC[k,n,c,L,K]]**	Let $c = S/K$ $P' = S' =$ $\mathcal{E}\frac{s(1+s)^{n-1}+Lsc(1+sc)^{n-1}}{(1+s)^n+L(1+sc)^{n-1}}$
	$P' = S' = \mathcal{E}\frac{\prod(1+a_j\overline{a_j})^n\prod s_j\prod(1+s_j+\overline{s_j})^{n-1}+L\prod(1+i_j)^n\prod(cs_j)\prod(1+cs_j+\overline{sj})^{n-1}}{\prod(1+a_j+\overline{a_j})^n\prod(1+s_j)^n+L\prod(1+i_j)^n\prod(1+cs_j+\overline{s_j})^{n-1}}$ where: $s_j = S_j/K_{S_j}$, $a_j = A_j/K_{A_j}$, $i_j = I_j/K_{I_j}$, $\overline{sj} = c\sum_k C_{jk}/K_{C_{jk}}$, $\overline{aj} = c\sum_k C_{jk}/K_{CA_{jk}}$	
	User Defined Arrows	
A↦B,USER[v,T,n,h,f] $\{$**P**$_1$**,P**$_2$**,..**$\}$**↦Q,USER[v,** $\{$**T**$_1$**,..**$\}$**,**$\{$**n**$_1$**,..**$\}$**,h,f]**	**[A|->B,USER[v,T,n,h,f]]** **[[P**$_1$**,P**$_2$**,..]|->Q,USER[v,** **[T,..],[n,..],h,f]]** **[[X**$_1$**,X**$_2$**,..]|->Y,mod[E],USER[v,** **[T,..],[n,..],h,f]]**[†] **[s**$_1$***A**$_1$**+s**$_2$***A**$_2$**+**$\cdots$**->q**$_1$***A**$_1$**+q**$_2$***A**$_2$**+**$\cdots$**,** **using[expr]]**[†,‡]	$B' = vf(h-TA^n)$ $Q' = vf(h-\sum_i T_iP_i^{n_i})$ $Y' = vEf(h-\sum_i T_iX_i^{n_i})$ $A'_i = (q_i - s_i)\times(\text{expr})$

[†]Only implemented in Python version. [‡]**expr** is any Python expression.

```
import numpy as np
from scipy.integrate import odeint

from math import *

foff = lambda t :-0.5*(t - 6000.0)*((t - 6000.0)**2 + 1)**(-0.5) + 0.5
fon = lambda t :(0.5*t - 374.5)*((t - 750.0)**2 + 1)**(-0.5) +  0.5

def ode_function_rhs(y,t):
  #
  # this odeint(..) compatible function was
  # automatically generated by Cellerator 2015-04-25 15:45:11
  # 2.7.8 (default, Oct 20 2014, 15:05:19)  [GCC 4.9.1]
  # linux2
  #
  # ============================================================
  # Model:
  #
  # #
  # # Phosphorylation Cascade
  # #
  # [K3 => K3p, mod[S], rates[a1,d1,k1]]
  # [K2 => K2p =>K2pp, mod[K3p], rates[a3,d3,k3]]
  # [K => Kp => Kpp, mod[K2pp], rates[a3,d3,k3]]
  # #
  # # competitive inhibition
  # #
  # [K3_S + Kpp <-> K3_S_Kpp, rates[a7, d7]]
  # #
  # # Dephosphorylation Cascade
  # #
  # [K3p => K3, mod[K3PH], rates[a4,d4,k4]]
  # [K2pp => K2p => K2, mod[K2PH], rates[a5,d5,k5]]
  # [Kpp => Kp => K, mod[KPH], rates[a6,d6,k6]]
  # #
  # # Input Signal
  # #
  # [Nil<->S, rates["a0*fon(t)*foff(t)", d0]]
  # ============================================================
  # rate constants
  a0 = 1.0
  a1 = 1.0
  a3 = 1.0
  a4 = 1.0
  a5 = 1.0
  a6 = 1.0
  a7 = 1.0
  d0 = 0.01
  d1 = 7.5
  d3 = 10.0
  d4 = 1.0
  d5 = 1.0
  d6 = 1.0
  d7 = 1.0
  k1 = 2.5
  k3 = 0.025
  k4 = 1.0
  k5 = 1.0
  k6 = 1.0
  stim_start = 750.0
  stim_stop = 6000.0
# pick up values from previous iteration
  K2p_K3p = max(0, y[0])
  K2p = max(0, y[1])
  K_K2pp = max(0, y[2])
  K3 = max(0, y[3])
  K2 = max(0, y[4])
  K2p_K2PH = max(0, y[5])
  K3p_K3PH = max(0, y[6])
  K3_S = max(0, y[7])
  K2pp = max(0, y[8])
  Kp_K2pp = max(0, y[9])
  Kp_KPH = max(0, y[10])
  Kpp = max(0, y[11])
  K3p = max(0, y[12])
  K = max(0, y[13])
  S = max(0, y[14])
  Kp = max(0, y[15])
  KPH = max(0, y[16])
  K3_S_Kpp = max(0, y[17])
  Kpp_KPH = max(0, y[18])
  K2_K3p = max(0, y[19])
  K2PH = max(0, y[20])
  K2pp_K2PH = max(0, y[21])
  K3PH = max(0, y[22])
# calculate derivatives of all variables
  yp=[0 for i in range(23)]
  yp[0] = K2p*K3p*a3 - K2p_K3p*d3 - K2p_K3p*k3
  yp[1] = -K2PH*K2p*a5 + K2_K3p*k3 - K2p*K3p*a3 + K2p_K2PH*d5 +
    K2p_K3p*d3 + K2pp_K2PH*k5
  yp[2] = K*K2pp*a3 - K_K2pp*d3 - K_K2pp*k3
  yp[3] = -K3*S*a1 + K3_S*d1 + K3p_K3PH*k4
  yp[4] = -K2*K3p*a3 + K2_K3p*d3 + K2p_K2PH*k5
  yp[5] = K2PH*K2p*a5 - K2p_K2PH*d5 - K2p_K2PH*k5
  yp[6] = K3PH*K3p*a4 - K3p_K3PH*d4 - K3p_K3PH*k4
  yp[7] = K3*S*a1 - K3_S*Kpp*a7 - K3_S*d1 - K3_S*k1 + K3_S_Kpp*d7
  yp[8] = -K*K2pp*a3 - K2PH*K2pp*a5 + K2p_K3p*k3 - K2pp*Kp*a3 +
    K2pp_K2PH*d5 + K_K2pp*d3 + K_K2pp*k3 + Kp_K2pp*d3 + Kp_K2pp*k3
  yp[9] = K2pp*Kp*a3 - Kp_K2pp*d3 - Kp_K2pp*k3
  yp[10] = KPH*Kp*a6 - Kp_KPH*d6 - Kp_KPH*k6
  yp[11] = -K3_S*Kpp*a7 + K3_S_Kpp*d7 - KPH*Kpp*a6 + Kp_K2pp*k3 +
    Kpp_KPH*d6
  yp[12] = -K2*K3p*a3 + K2_K3p*d3 + K2_K3p*k3 - K2p*K3p*a3 + K2p_K3p*d3
   + K2p_K3p*k3 - K3PH*K3p*a4 + K3_S*k1 + K3p_K3PH*d4
  yp[13] = -K*K2pp*a3 + K_K2pp*d3 + Kp_KPH*k6
  yp[14] = -K3*S*a1 + K3_S*d1 + K3_S*k1 - S*d0 + a0*foff(t)*fon(t)
  yp[15] = -K2pp*Kp*a3 - KPH*Kp*a6 + K_K2pp*k3 + Kp_K2pp*d3 + Kp_KPH*d6
   + Kpp_KPH*k6
  yp[16] = -KPH*Kp*a6 - KPH*Kpp*a6 + Kp_KPH*d6 + Kp_KPH*k6 + Kpp_KPH*d6
   + Kpp_KPH*k6
  yp[17] = K3_S*Kpp*a7 - K3_S_Kpp*d7
  yp[18] = KPH*Kpp*a6 - Kpp_KPH*d6 - Kpp_KPH*k6
  yp[19] = K2*K3p*a3 - K2_K3p*d3 - K2_K3p*k3
  yp[20] = -K2PH*K2p*a5 - K2PH*K2pp*a5 + K2p_K2PH*d5 + K2p_K2PH*k5 +
    K2pp_K2PH*d5 + K2pp_K2PH*k5
  yp[21] = K2PH*K2pp*a5 - K2pp_K2PH*d5 - K2pp_K2PH*k5
  yp[22] = -K3PH*K3p*a4 + K3p_K3PH*d4 + K3p_K3PH*k4
  return yp



def thesolver():
    filename ="/home/mathman/Desktop/ipypaper/MAPK/MAPK.model"
    variables=[’K2p_K3p’, ’K2p’, ’K_K2pp’, ’K3’, ’K2’, ’K2p_K2PH’,
     ’K3p_K3PH’, ’K3_S’, ’K2pp’, ’Kp_K2pp’, ’Kp_KPH’, ’Kpp’, ’K3p’,
      ’K’, ’S’, ’Kp’, ’KPH’, ’K3_S_Kpp’, ’Kpp_KPH’, ’K2_K3p’, ’K2PH’,
      ’K2pp_K2PH’, ’K3PH’]
    runtime = 12000
    stepsize = 10
    t = np.arange(0,runtime+stepsize,stepsize)
    y0 = [’0’, 0.0, ’0’, 100.0, 300.0, ’0’, ’0’, ’0’, 0.0, ’0’, ’0’,
     0.0, 0.0, 300.0, 1.0, 0.0, 1.0, ’0’, ’0’, ’0’, 1.0, ’0’, 10.0]
    sol = odeint(ode_function_rhs, y0, t, mxstep=50000)
    return sol

if __name__=="__main__":
    thesolver()
```

Fig. S4. Auto-generated code for the model shown in fig. 1 of the main text of the article. The only dependencies of this code are numpy and scipy; the user could hypothetically run this program from the command line, e.g, as `Python mysolver.py`, by integrating the code into another program, or using it in iPython.